Ripgrep lowargs.rs: The Raw Flag Mirror

What This File Does

The lowargs.rs file defines LowArgs and its supporting types — a direct mirror of command-line flags. Where HiArgs represents computed, validated configuration, LowArgs represents exactly what the user typed. No heuristics, no derived values, no environment probing.

The file is roughly 720 lines, but most of that is type definitions. It's essentially a schema describing every flag ripgrep accepts and the valid values each can take.

Section 1: The Design Philosophy

The module doc comment articulates a key constraint: "populating low level arguments should not require anything other than validating what the user has provided." LowArgs won't contain a hostname for hyperlinks because discovering the hostname requires syscalls or running binaries. That work belongs in HiArgs.

This separation serves robustness. Parsing LowArgs can fail if flags have invalid values, but those failures are predictable and local. HiArgs transformation can fail for environmental reasons — missing files, inaccessible directories, unsupported platform features. By separating these concerns, ripgrep ensures --help always works even when the environment is broken.

The practical benefit: you can always get help even if your current directory was deleted, your config file is corrupted, or you're on a weird platform where hostname detection fails.

See: Companion Code Section 1

Section 2: The LowArgs Struct

The struct contains roughly 70 fields, one for each conceptual flag or flag group. Fields are public within the crate but not externally visible. The parser populates them; HiArgs reads them.

Fields group into categories. Essential arguments include the mode, positional arguments, and pattern sources. Display options cover colors, headings, line numbers. Search options handle case

sensitivity, word boundaries, multiline matching. Filter options control ignore rules, file types, and globs. Performance options set thread counts, memory limits, and mmap behavior.

The Default derive provides sensible starting values. Most booleans start false. Most Options start None. Enums typically default to their "auto" or "standard" variant.

See: Companion Code Section 2

Section 3: SpecialMode — The Short-Circuit Cases

SpecialMode represents flags that bypass normal processing entirely. When present, ripgrep shows the requested information and exits immediately.

Five variants exist: HelpShort (-h), HelpLong (--help), VersionShort (-V), VersionLong (--version), and VersionPCRE2 (--pcre2-version). The short versus long distinction matters because they produce different output formats.

The comment explains why these get special treatment: "we avoid converting low-level argument types into higher level arguments types that can fail for various reasons related to the environment." Help should always work, even when nothing else does.

See: Companion Code Section 3

Section 4: Mode — What Ripgrep Should Do

Mode represents the primary operation ripgrep will perform. Four variants cover the cases: Search (find matches), Files (list files that would be searched), Types (list file type definitions), and Generate (produce man pages or completions).

Search wraps a SearchMode that further distinguishes standard output, files-with-matches, counts, and JSON. This nesting reflects that most search variations share infrastructure while Generate modes are completely different code paths.

The update method implements override semantics. Once in a non-search mode, search modes can't override it. So "rg --files -l pattern" stays in Files mode despite -l normally enabling files-with-matches. This matches user expectations — explicit mode selection takes precedence.

See: Companion Code Section 4

Section 5: SearchMode — Search Output Variations

SearchMode distinguishes how search results appear. Standard prints matching lines. FilesWithMatches prints only filenames containing matches. FilesWithoutMatch prints filenames without matches. Count prints match counts per file. CountMatches counts individual matches rather than matching lines. JSON outputs structured data.

There's no explicit flag for Standard — it's the default. Other modes have both enabling flags (-l, -c, --json) and negation flags (--no-json) to return to Standard.

The distinction between Count and CountMatches matters for patterns that match multiple times per line. Count reports one match per matching line. CountMatches reports the actual match count.

See: Companion Code Section 5

Section 6: BinaryMode — Handling Non-Text Files

BinaryMode controls how ripgrep treats files containing binary data (specifically NUL bytes, which don't appear in text files).

Auto applies different strategies based on how the file was specified. Explicitly named files get searched with binary data converted or suppressed. Implicitly discovered files during traversal get skipped entirely when binary content is detected.

SearchAndSuppress searches the file but replaces NUL bytes with line terminators and suppresses match output, showing only a warning. This prevents memory issues from extremely long "lines" in binary files.

AsText (-a/--text) treats everything as text, disabling all binary detection. Useful when you know a file contains mostly text despite some binary content.

See: Companion Code Section 6

Section 7: CaseMode — Pattern Matching Sensitivity

CaseMode controls whether pattern matching distinguishes uppercase from lowercase.

Sensitive is the default — 'a' matches only 'a'. Insensitive (-i) makes 'a' match both 'a' and 'A'. Smart (-S) enables case-insensitive matching only when the pattern contains no uppercase characters. The pattern 'foo' matches 'FOO', but 'Foo' matches only 'Foo'.

Smart case is particularly useful for interactive use. Searching for a common word doesn't require caring about case. Searching for a specific CamelCase identifier naturally becomes case-sensitive.

See: Companion Code Section 7

Section 8: ColorChoice — Output Coloring

ColorChoice controls when ripgrep produces colored output with ANSI escape codes.

Never disables colors entirely. Always enables them unconditionally. Auto (the default) enables colors only when stdout is a terminal. Ansi forces ANSI escapes even on Windows where ripgrep might otherwise use legacy console APIs.

The to_termcolor method converts to the termcolor crate's equivalent enum. This conversion happens frequently since termcolor handles the actual escape code generation.

See: Companion Code Section 8

Section 9: ContextMode — Lines Around Matches

ContextMode controls how many non-matching lines appear around each match for context.

Passthru (-p/--passthru) shows all lines, with matches highlighted. This is useful for viewing entire files with matches marked.

Limited specifies exact counts via ContextModeLimited, which tracks before, after, and both separately. The -B flag sets before context, -A sets after context, -C sets both. The separation matters because -B and -A always override -C regardless of flag order.

The get method in ContextModeLimited implements the precedence rules. If both is set to 5 but before is set to 2, you get 2 lines before and 5 lines after.

See: Companion Code Section 9

Section 10: EngineChoice — Regex Implementation

EngineChoice selects which regex engine processes patterns.

Default uses Rust's regex crate (technically regex-automata). It's fast and has good error messages but lacks some advanced features.

PCRE2 uses the PCRE2 library, supporting lookaround assertions and backreferences. It's available only when ripgrep is compiled with the pcre2 feature.

Auto tries Default first, falling back to PCRE2 if compilation fails. This lets patterns requiring advanced features work without explicit engine selection.

See: Companion Code Section 10

Section 11: MmapMode — Memory Mapping Strategy

MmapMode controls whether ripgrep uses memory-mapped I/O for reading files.

Auto (default) uses heuristics. Memory maps work well for a few large files but have overhead for many small files. The heuristic considers file count and whether all inputs are regular files.

AlwaysTryMmap forces memory mapping when possible. Some inputs can't be mapped (stdin, FIFOs), but regular files will use mmap.

Never disables memory mapping entirely. Even multiline mode, which needs the entire file in memory, will read into heap-allocated buffers instead.

See: Companion Code Section 11

Section 12: PatternSource — Where Patterns Come From

PatternSource tracks pattern origins. Patterns can come from -e/--regexp flags (Regexp variant) or from files via -f/--file (File variant).

Why track the source? Order matters. Patterns from multiple sources are searched in the order provided. Also, patterns from files require reading those files, which might fail. The error message should indicate which file caused the problem.

The File variant stores the path. The Regexp variant stores the pattern string directly. Both get processed during the LowArgs to HiArgs transformation.

See: Companion Code Section 12

Section 13: SortMode — Result Ordering

SortMode controls file processing order. The struct combines a sorting criterion (SortModeKind) with a reversal flag.

Four criteria exist: Path (alphabetical), LastModified, LastAccessed, and Created. Not all platforms support all criteria — the supported method checks by probing the current executable's metadata.

When sorting is enabled, parallelism is disabled. You can't sort results that arrive out of order. Path sorting ascending gets special optimization — the walker can sort during traversal.

See: Companion Code Section 13

Section 14: TypeChange — File Type Modifications

TypeChange represents modifications to file type definitions. Ripgrep has built-in types (rust, python, c, etc.) that users can modify.

Clear removes a type definition entirely. Add creates a new definition with name and glob pattern. Select enables a type for filtering — only files matching that type get searched. Negate inverts a type — files matching are excluded.

These accumulate in a Vec because order matters. You might clear all types, add custom definitions, then select which to use. The Vec preserves that sequence for HiArgs to process.

See: Companion Code Section 14

Section 15: Separator Types

Several types represent customizable separators with escape sequence support.

ContextSeparator (default "--") appears between non-contiguous context blocks.
FieldContextSeparator (default "-") separates metadata fields on context lines.
FieldMatchSeparator (default ":") separates metadata fields on match lines.

Each provides a new constructor that handles escape sequence unescaping. Users can specify "\t" for tabs, "\x00" for NUL bytes, etc. The BString type (from the bstr crate) holds arbitrary bytes, not just valid UTF-8.

See: Companion Code Section 15

Key Takeaways

First, LowArgs is a direct mirror of CLI flags. No computation, no heuristics, no environmental queries.

Second, the separation between LowArgs and HiArgs isolates parsing failures from environmental failures. This ensures help always works.

Third, enum types with explicit variants make flag interactions clear. CaseMode::Smart is a distinct state, not a combination of other flags.

Fourth, order-sensitive operations (patterns, type changes) use Vec to preserve user intent.

Fifth, custom types handle validation at parse time. Invalid separator escapes fail immediately, not during search.

What to Read Next

Understanding lowargs.rs completes the argument story:

How does the parser populate LowArgs? Read flags/parse.rs.

How does HiArgs transform these raw values? Read flags/hiargs.rs (already covered).

How are individual flags defined? Read flags/defs.rs.